REMARKS

Reconsideration of this application is respectfully requested in view of the foregoing

amendment and the following remarks.

Claims 16-65 remain pending in this application. Claim 49 has been amended to correct

matters of form. No other amendments have been made to the claims.

The only claim rejection made against pending claims 52-64 is an obviousness-type

double patenting rejection. Filed concurrently herewith is a Terminal Disclaimer with respect to

Applicants' granted U.S. Patent 6,696,631. Claims 52-64 should therefore be immediately

allowable. For the reasons stated below, Applicant respectfully submits that all of the other

claims pending in this application are also in condition for allowance.

In the Office Action, claims 24, 28, 35, 38, 43, 45 and 49 were rejected under 35 U.S.C.

§112, second paragraph; claims 16-24, 26, 27, 35-37 and 39 were rejected under 35 U.S.C.

§102(b) as being anticipated by U.S. Patent 5,852,251 to Su et al ("SU"); claim 65 was rejected

under 35 U.S.C. §102(b) as being anticipated by Cakewalk Professional for Windows User's

Manaul ("Cakewalk");claims 25, 38, 40-42, 44-47 and 50-51 were rejected under 35 U.S.C.

§103(a) as being unpatentable over Su in view of Cakewalk; claims 26, 28-34, 43, 48 and 49

were rejected under 35 U.S.C. §103(a) as being unpatentable over Su in view of U.S. Patent

5,693,903 to Heidorn et al ("Heidorn"); and claims 52-64 were rejected on the ground of

obviousness-type double patenting over claims 21-26 and 28-34 of U.S. Patent 6,696,631. These

grounds of rejection are respectfully traversed.

The present invention is directed to a unique music performance system. As explained in the background section of the present application, due to, for example, economics and space, the size of orchestras for performances has been reduced. Unfortunately, when such a reduction takes place the overall quality of the music suffers. While there have been numerous attempts to address the resulting shortcomings of reduced orchestra sizes, none has been able to adequately compensate for the missing orchestra pieces or the ability of individual musicians in the orchestra to modify a particular performance as desired by a musical director or as necessitated by a particular venue. The presently claimed invention provides a unique appraoch for overcoming the shortcomings of prior systems.

I.   §112, SECOND PARAGRAPH REJECTIONS

A.   Claim 24: "the number of times an event is encountered"

Within the scope of the invention, every data structure can be broken down into individual events. Examples of such events are digital representations of notes, of volume values, of metaevents that influence interpretation of regular events, etc. All events are performed in series. Often, certain parts of the performance will be repeated again. For example, there may be an eight bar introduction to a musical piece. The composer, in the original notation, instead of writing out this section twice, will instead use repeat markings to indicate that the musicians interpreting the score should go back and play those eight measures a second time. In this instance, every musical note within those measures is performed twice. Other versions of the same piece may require that this section is repeated more times, fewer times, or not at all. Each of these possibilities creates a situation where a particular event is encountered a different number of times. This type of activity is duplicated within the present invention. Since

every performance may rely on different numbers of repetitions, it becomes essential to be able

to vary the number of times an event is encountered within the context of the performance. These

instructions would be written in the second data structure.

In addition, certain types of events may want to be ignored on certain repetitions of these

areas. For example, there may be a note that is played by the flute, and this note should only be

played the last time that the section is to be repeated. In this case, control information is required

such that when this particular note is encountered within the section, it is ignored unless the

section is in the final repeat.

Other possibilities include notes that should be performed every other time, notes that

should be performed only the first time, etc.

Since each section is only stored once in the first data structure, it is the second data

structure that determines the progression through these sections. As a result, many events may

be encountered multiple times in a single performance.

In connection with the foregoing, U.S. Patent 6,696,631 at col. 12, line 58 (hereafter,

'631, 12:58) defines a navigation map, which is the data object that determines which sections

are repeated, and how often.

'631, 24:46 defines the wait state, which indicates that a particular event should not be

activated until the requisite number of encounters has occurred.

'631, 24:51 defines the times state, which specifies the number of times that an event will

be activated on encountered before it is ignored.

'631, 25:2, and more specifically '631, 25:45 discusses the pattern argument, that could

be used to activate or deactivate a particular event depending on a pattern of activation and

quiescence.

'631, 28:45 discusses the relocate type events that allow random access movement within

the data structure such that these events can be encountered multiple times.

B.   Claim 28: "resultant map"

Mapping and maps are crucial concepts in the present invention. Because the invention

supports multiple maps for any particular parameter, it is possible to store multiple maps and

from these to use a variety of combining operations to determine a single result.

Since one way that a user can create a map is to record a set of performance

modifications during rehearsal, and since this can be stored as a map, and since the user can then

record another map during rehearsal or performance into a different map, and since these two

maps can be combined into a map group, and then the result of this map group can be a single set

of results, claim 28 should therefore become clear.

For a detailed explanation of "how a plurality of maps can be weighted and averaged to

yield a composite" map, see the following:

'631, 17:30 and following discusses maps.

'631, 17:35 and following discusses recording performance information from external

control into data structures defined as maps.

'631, 20:16 discusses different ways that these maps can be created. Technique #2 is

recording from the output stream during rehearsal or performance.

'631, 18:36 – 19:64 discuss in detail how multiple maps can be combined into a map group, and how the result of this map group is a single result that is used to affect the underlying data structure.

'631, 21:48 discusses how to take multiple maps and collapse them into a single resultant map.

C.   Claim 35: Definition of "Entity"

The examiner is correct in that the term "entity" refers to the user. In certain cases, the user is defined as a producing organization, such as a school, theatre company, etc. The structure is then the hierarchical organization being referred to. Applicants are willing to amend this claim if the Examiner believes this would place the claim in better condition for issuance.

D.   Claim 38: "Tap Release Velocity"

"Tap" is a term that specifically applies to how the performer creates tempo (or speed) variation by applied precisely timed control information to the invention. Tap tempo is a standard feature of many applications, and should be considered a standard definition. However, the scope of the present invention extends tap capabilities in a number of ways unanticipated by other inventions or prior patents.

Once such non-anticipation is the "tap release" of claim 38. This tap is derived from a performer who is playing from an external source, and thereby creating tap actions. These can come from a variety of sources, as indicated by '631, 23:24. Certain types of communication, such as MIDI from a controller keyboard, support an action that sends an event when a key is depressed, and another action when that same key is released. When the key is released, many instruments generate a note-off event as defined in the MIDI specification 1.0. This note off

event contains a velocity value that is determined by how quickly the finger is removed from the

key. This is the action that determines a tap release velocity. A typical musician would find that

this type of control can be useful if one were to want to apply information of the decay time (the

length of time that the sound takes to go away) of a defined set of instruments, for example.

Since the tap release velocity is not sent directly out the output stream, but is instead used to

modify the performance, this can be reinterpreted based on the second data structure, and this in

turn can be applied to any number of output events.

E.   Claim 43: "Exiting a Vamp"

Whereas turning off the power would certainly "exit the vamp immediately" as suggested

by the Examiner, those skilled in the art of music would understand the term "exiting a vamp" to

require continuing ahead in the performance. A reasonable definition of vamp can be found in

the Wikipedia web site.

> Similarly, in musical theater, a vamp is a figure of one or more measures which the orchestra
> repeats during dialogue or stage business. Here the purpose of a vamp is to allow the singers as
> much time to prepare for the song or the next verse as is necessary, without either requiring the
> music to pause until the singers are ready or requiring the action on stage to be carefully
> synchronized with music of a fixed length. Once the vamp section is completed, the music will
> continue on to the next section, and create a continuous underscoring.
> http://en.wikipedia.org/wiki/Vamp_%28music%29 06-16-06.

Because the number of times that a vamp section is to be repeated is indeterminate, it is

necessary to signal the invention when the intention is to stop repeating that section. This is

called "exiting a vamp." It is not enough to simply turn off the power, as it would not be enough

for an acoustic orchestra to simply stop playing. The detailed specification clearly states that a

vamp is a type of relocate event. See, e.g., '631, 28:45. Relocate is an action type that is stored in the second data structure, and determines how to navigate through the performance in ways that are nonlinear: in other words, jumping backwards and forwards through the performance. Vamp is a specific subset of the relocate event, and allows for an indeterminate number of repetitions of the defined musical region. Since the number of times is indeterminate, theoretically, the vamped section could continue without exit forever. It thus requires an explicit command from an external source in order to exit the vamp and continue into the next section.

A Vamp event is defined in the specification at '631, 45:40.

A detailed explanation of exiting a vamp can be found at '631, 35:31.

An immediate exit of a vamp can be implemented by defining a key that will be programmed to relocate to the appropriate location. Since a key can be dynamically allocated so that it can be responsible for a variety of actions using the external control map '631, 18:21, this easily allows the user to create a way of immediately exiting the vamp, as recited in claim 43.

F.   Claim 45: Arbitrary measure numbers

In musical theatre, and other performances, the first measure of a song may not always be labeled as "1". In fact, due to rearrangements and modifications to the score during the rehearsal process, many scores reflect strange orders and measure number values. Since these are the measure numbers referred to by the musical director and musicians, it becomes essential that the invention is able to refer to these nonlinear measure numbers. This requirement is specified at '631, 10:28 to 10:43.

Because these measure numbers need to be tied to information in the first data structure, which in the preferred embodiment is a standard MIDI file, it is necessary to develop a procedure

that allows for this type of identification. This is clearly outlined in '631, 8:45 to 8:65. Measure

numbers have meaning beyond merely their location at the front of each measure. For example,

any beat or subdivision of the beat within that measure would also have to be referred to by the

renamed or arbitrarily named measure number. Thus the invention would have to be able to refer

to a particular location as measure 12B beat 3.

This information is extracted from the first data structure at load time, and stored in a

metric map (see, e.g., '631, 12:52 and particularly 20:8).

G.   Claim 49: Correct Tap subdivision within a performance

Claim 49 has been amended to correct a minor matter of form. It is noted that a tap

subdivision relates to tap performance, where the user is able to move the invention forward by

certain increments of the metric structure. This is clearly defined at '631, 25:36. Because the

nature of the invention allows for multiple versions of tap subdivisions, and different performers

may want to use different subdivisions for the same section, it is important to be able to allow for

this. Although individual tap subdivisions can be placed into the score using the editing features,

often long stretches of the piece may have a particular series of different tap values that repeat

themselves periodically. In this case, it becomes easier to program one single pattern instead of

having to input many different subdivisions again and again. Because a pattern can be initiated

at any time, there is no way of determining a correct current subdivision without referring to the

pattern definitions.

A set of defined patterns can be stored in a definition file, and then the patterns of

subdivisions can be stored in the metric map. The metric map is part of the second data

structure.

A very detailed explanation about how the system uses tap and tap subdivisions can be found at '631, 29:7 to 31:43.

II.   §102 REJECTION BASED ON SU

A.   Independent Claim 16

The present invention provides a system that accesses a first data structure representing a plurality of musical pieces. Su has no structure that allows for multiple pieces, and in fact, it is impossible to use the Su architecture so that manipulation of individual pieces, as well as manipulation of all pieces, can occur. Because Su cannot anticipate or differentiate between multiple pieces, it can not provide information in the second data structure including instructions for selecting from among and arranging the plurality of musical pieces including arranging music on the respective tracks by applying the second data structure to the first data structure to produce the musical performance. This inability to differentiate between separate musical pieces, or even to support separate pieces, results in Su being unable to provide the stated purpose of the present invention, which is to support live performances of entire shows.

Trying to use Su to anticipate multiple pieces by creating multiple instances of the single Su device will fail, because there is no mechanism of coordinating the multiple devices such that the order of pieces can be automated, or that individual instruments or parts can be modified with a single command. To do so would require an overarching structure that is not anticipated by the Su invention. This structure and ability to coordinate multiple pieces, including rearranging said pieces, is crucial to the scope of the invention.

Su et al. discloses a system that takes any standard MIDI file 418 received by MIDI controller 406 and immediately converts it to a Type 0 MIDI file, which combines all tracks into

a single track. This inherently precludes any support of instruments numbering greater than 16.

The current invention, on the other hand, can support an arbitrary number of tracks. Because of

the nature of modern musical performance, it is desirable that musical structures of greater than

16 instruments be supported. Indeed, it is not uncommon to have instrumentations that exceed

60 instruments.

In the same manner that using multiple instances of the Su invention will fail to provide

the necessary control of multiple pieces, multiple instances of Su will also fail to support greater

than 16 tracks of instrument. Because it is often necessary to provide global control of all or a

majority of instruments, and since instructions about how this control is to be interpreted must be

programmed, an additional overarching structure is essential to correctly interpret how second

data structure instructions will be applied to the first data structure. Su does not disclose such an

overarching structure.

In fact, many capabilities of the current invention are impossible to be anticipated by Su,

While Su is directed to real-time dynamic MIDI control, this reference fails to disclose or even

suggest features of the claimed invention. Specifically, Su does not disclose the control of music

from a plurality of instruments stored on respective tracks, since Su uses as its first step the

conversion of Type 1 and Type 2 MIDI files 418 into Type 0 MIDI files. Type 0 MIDI files

convert all separate tracks into a single track, and individuation of instruments is achieved by

reading the channel number associated with a particular event, and then modifying that

information based on a channel number. Since Type 1 and Type 2 files are able to disassociate

different tracks assigned to the same channel, many required manipulations able to be performed

on a type 1 file (which is what is used by the present invention) are not able to be performed on a

type 0 file.

The discussion of global definitions that apply to greater than one song can be found at

'631, 13:56.

Su also does not disclose "<u>selecting from among and arranging</u>" a plurality of musical

pieces as recited in claim 16. The schedule control processor 800 of Su does not disclose this

functionality.

Applicants submit that not only is independent claim 16 patentable over Su, but also that

several of the claims dependent on claim 16 are clearly distinguishable from the teachings of Su.

B.   Dependent claims 23 and 27 - mapping and maps

There are numerous times in the patent application and within the claims that the

Applicant recites the use of maps. At times the examiner considers them but rejects the claim by

using the term "mapping". It appears to the Applicant that there is some confusion about how

the term "map" is used in the instant application.

When data is "mapped" there is a modification between the data state and the output

state. This certainly occurs in many related inventions, and if the application claim were to stand

or fall merely on this capability, then the claim would fail. However, the term "map" is used in

all cases as a noun, as opposed to the verb "to map". Each map is a data structure that has a

specific technical meaning within the scope of the invention. the set of maps determines how the

mapping will occur within the performance. Many related inventions have ways of determining

how these mappings occur, but do not provide the flexibility of the mapping structures as

detailed in the instant application.

For example, Su discloses a "Schedule control processor 800" (Su at 6:59) that allows

modification of how the administrator interprets the MIDI data. One can program this such that

certain parameters can be changed at different times in the piece. However, Su does not

anticipate the capabilities of maps as defined in the present invention. There is no way to isolate

different types of parameters. The Schedule control processor contains all different types of

information. There is no provision for generating multiple maps that affect the same parameter

and are stored as map groups. There is no way of generating a navigation structure such that the

performance can move arbitrarily through the data structure. There are no tap patterns or hot key

capabilities.

The extraction of actions and placement into maps is discussed at '631, 12:41.

The map storage disclosure for the current invention can be found at '631, 17:30 onward.

The map group can be found at '631, 18:39. Techniques for combining, masking, merging,

averaging, etc., these maps can be found starting at '631, 18:42 and continuing to '631, 20:8.

Su does not disclose the claimed limitations.

C. Dependent claim 22

In any performance, there are a number of unique events. To look at a particular score,

one might find that a violinist will be asked to play a number of middle c's. Even though each

middle c is the same pitch, each is also notated as a separate event. In other words, an event can

be identified not only by its pitch, but by its location within the score. Thus, middle c at measure

4 beat 1 is a different event from a middle c at measure 5 beat 2.

Sometimes this information can be determined beforehand, and then these repetitions etc

can be preprogrammed. '631, 13:40 discusses how the invention anticipates interaction between

the music director and the programmer, such that this type of programming is necessary within the scope of the invention.

In a data structure of a score, these individual events are stored in a file. In some cases, individual events will be encountered more than once. For example, if there is a repeated section, the same events will be encountered each time the section is repeated. Depending on the performance, these sections can be of multiple or indeterminate lengths. It becomes necessary to program using the second data structure.

When the event is a Metaevent or action, then that event can be programmed with a times field (the number of times that the event is to be activated upon being encountered), and a wait field (the number of times that the event is to remain quiescent upon being encountered). These definitions relate to claim 24, and the details can be found at '631, 24:51 and following. Su is silent on this feature.

D.   Dependent claim 27

Note that as per the discussion above, layered maps are separate mapping entities that are combined into a map group. This map group results in being able to take multiple sets of different mappings of the same parameter, and combine them into a single modification schema. For example, two different maps with different volume modification information can be combined to yield one result. These composite groups are discussed at '631, 18:19. Su does not disclose this feature of the claimed invention.

III.   §102(B) REJECTION BASED ON CAKEWALK

Although Cakewalk, and for that matter, most other sequencer applications, allow an instrument or track to be assigned to an output port, this information is stored within a

proprietary format, and not in a standard MIDI file. A Cakewalk file does not conform to a

Standard MIDI file, and, if converted, the port information will be lost. This is because the

MIDI specification was created before the idea that multiple ports would be a useful feature. In

addition, each channel message is limited a 4 bit component that defines MIDI channel. There is

no way of embedding this information within the standard MIDI specification.

Since standard MIDI files are important conduits of sequenced MIDI data between

disparate applications, it becomes important to be able to associate a port ID along with a track.

The present invention bypasses this by using a preferred embodiment syntax that is embedded in

the track name (so that a track labeled A_flute would reference instrument "flute" and go out the

A port, and another track called B_oboe would reference instrument "oboe" and send its

information out the B port). In this way, port information may be stored within the standard

MIDI file, allowing for port differentiation within the existing standard MID file format. Any

application that can parse this information can then take advantage of port information.

This information can be found in the specification at '631, 9:65. Also, reference is made

to Figure 4 which demonstrates how the port information is bound to the track name in the

standard MIDI file.

In view of the foregoing, the Cakewalk disclosure does not anticipate the limitations of

independent claim 65.

IV. §103 REJECTION BASED ON SU AND CAKEWALK

A number of claims have been rejected based on the capabilities of Cakewalk, which is a

standard sequencer. Sequencers are capable of a significant number of powerful operations, and

in fact the inventors relied on standard sequencers for a number of years early in their

experimentation. If sequencers were able to accomplish a number of additional tasks, then it would have been unnecessary to invent more powerful control capabilities. However, this is not the case, and results in the current scope of invention. Whereas the current scope of invention relies on sequencing techniques as one of the components of the invention, there is no attempt to patent capabilities found in standard sequencer technologies. In fact, all claims cited are specifically targeted at capabilities that are unanticipated by sequencers.

Below are several significant ways in which the scope of the present invention extends to capabilities entirely unanticipated by sequencer applications.

A) Data storage

Sequencers function as recording devices capable of storing a vast amount of information. This information is used to control external synthesizers or other sound modules. In order to modify how that external sound module is controlled, the information stored in the sequencer is directly modified. Note that this means the actual stored MIDI data is changed. This creates several problems in live performance environments.

1) Data can become forked.

In order to make a change, direct modification of the data file occurs. Two separate individuals would therefore end up with different copies of the underlying data. If a mistake in the score was discovered by one individual, the other individual would be unable to take advantage of the correction without specifically entering that modification into their copy. A detailed description of any such modification would have to be made, and strict compliance rules set in order to guarantee data integrity. Since score data is interspersed with show data, and

individual 1's show data will be different from individual 2's, it would be impossible to

synchronize by transferring the modified files from one production to the next.

2) Original pristine data is lost.

If a particular production requires a modification to the score, then the underlying data

must be modified. For example, perhaps the director would like measures 1 through 10 of song

1 to be skipped. The programmer would then delete this portion of the music. If at a later time

the director wanted to restore that information, it would be impossible without searching

backwards through multiple backup copies. If the backup copy had preceded additional

modifications, then the restoration of the original measures would also eliminate the later

modifications. This creates a situation where quick revisions become impossible.

In the present invention, integrity of the underlying data is maintained at all times. By

creating distortions to the underlying data, and storing how these distortions will affect the

underlying data (first data structure) in a second data structure, the ability to perform significant

modifications to a performance without affecting the underlying data is maintained. If a mistake

is discovered, or if the composer wants to add or remove musical information, then the pristine

data can be changed to reflect the correction or emendation, and then this file can be published to

all productions and performance versions without losing the show specific information stored in

the second data structure.

B) Show Structure

By their nature, sequencers are designed to work on one song at a time. In contrast, most

productions consist of a compilation of multiple songs (plurality of songs). Although some

sequencers have the ability to store multiple sequences (songs) within a single file, each

sequence is treated as a separate entity. No provision is made to provide an overall data structure

that allows global changes in instruments that can apply to each separate sequence. So for

example, if the musical director desires to mute the flute part for the entire show, the

programmer of a sequencer would have to go to each song, and delete the flute information

individually. In a show with forty such songs (and this is a very common number of songs), the

operation is tedious. Multiply these requests by a normal number of such modifications, and this

type of operation becomes prohibitive.

Another example of how the current scope of invention can manipulate show structures

in ways unanticipated by sequencers can be seen in song arrangement. Very often, the

performance requires that a piece of music be repeated or rearranged. Perhaps there is an

interlude, for example, that was originally intended to be used between songs 2 and 3. The

director, however, would also like to use the same interlude between songs 5 and 6. In a

sequencer, one would have to create a new copy of that sequence, and then place it in the

appropriate order. Or, one would have to define a specific command to refer to each sequence,

and then activate that command to move to that sequence. In the first instance, by creating a

copy, two different instances of the sequence exist, and the user runs into the problem of forked

data. In the latter instance, there is not a single ordering of sequences such that the integrity of

the show order can be maintained, since the operator would have to explicitly hit different keys

to activate different sequences, instead of relying on a "next sequence" single key. Either

"solution" does not anticipate many of the features that the present invention is able to implement.

In the present invention, by maintaining a show file that refers to all songs in a particular complete show, the user is able to dictate changes that can affect each song in the show, is able to maintain a list of songs that can reference individual sequences multiple times and easily maintain a sequential ordering of said sequences, all without modifying the underlying sequence structures.

C) Instrument Structures

In a sequencer, although each track can be labeled with a name, the actual mechanism of imparting MIDI information is determined by the channel and port assignment that is assigned to that track. The instrument name then becomes merely a label that allows the user to identify the type of data stored on a particular track. Changing the name of the track does not affect how that track is interpreted, and different instances (sequences) with identical track names are not bound together.

In the present invention, the ability of the second data structure to contain instrument definitions allows these bindings and definitions to occur. As a result, many types of global or local operations can be implemented that are unanticipated by the sequencers. For example, an instrument definition may contain information regarding how loud a particular instrument is to be played. Perhaps the user has decided that an instrument should play at 50% of the volume defined by the underlying sequences (first data structure). By modifying a single definition in the show file (second data structure), this instrument is modified in each song, and thereby in

each underlying sequence. Thus multiple sequences have been modified with a single command, and without modifying the underlying sequence's data. In addition, if the user were to go to the underlying sequence, and change the name of a track to the instrument referred to above, that track would now be modified in the same way that the other tracks were.

D) Additional structures

There are many additional structures that differ from the way that traditional sequencers implement data organization, and they could be explicated here, but the basic concept of <u>a second data structure that modifies the output without affecting the first data structure, and can synchronize across multiple songs</u> is the underlying principle that creates these capabilities.

With this background discussion, several of the dependent claims rejected based on a combination of Su and Cakewalk are discussed below.

A.   Dependent claim 25

Although Cakewalk supports certain types of external control, claim 25 references the ability to control the number of times that a particular event is activated upon being encountered. There is no provision in Cakewalk for this type of behavior. See above for a discussion about "wait" and "times" values.

B.   Dependent claim 38

Although the examiner states that tap release velocity is functionally equivalent to after-touch control, this is not the case. Note release velocity is associated with a note-off event, and will generate a value based on the speed at which the key is released. This is a single value event.

The syntax is as follows:

400413287v1

1000nnnn 0nnnnnnn 0vvvvvvv

The third byte is release velocity.

After-touch is sent when the amount of pressure applied to a key changes. This means that many events will be sent during the duration of a held note. Also, the value will always drop to 0 before the key is released, and no values will be sent after this time. Thus aftertouch always occurs **during** a note, and release velocity will always occur **after** a note.

The syntax of polyphonic aftertouch is as follows:

1010nnn 0nnnnnnn 0ppppppp

In addition, Cakewalk will happily record the polyphonic aftertouch, but this in no way can be used to modify the output of the MIDI file. It instead is stored as a discrete event that is then blindly sent on to the sequencer output during playback. So even though Cakewalk can affect the output of an instrument, there is no provision for it to be applied to an instrument property or properties <u>such that the resultant musical output of a plurality of instruments is modified.</u>

The underlying differentiation between the Cakewalk system and the scope of the present invention is that in Cakewalk external recorded events are designed to be stored directly into sequence tracks, and then to be played back at a future time. The current invention accepts input that can dynamically modify multiple properties of a plurality of instruments and performances without affecting the underlying data.

C.   Dependent claim 40

Although Cakewalk can store markers, and in fact the structure of standard MIDI files is designed to allow for markers, there is no functionality in the markers other than as placeholders.

In the present invention, these markers can be interpreted as metaevents, which are used to control a variety of different ways that the first data structure can be manipulated.

The difference between how Cakewalk interprets metaevents and the present invention is significant. In the case of Cakewalk, each marker would become a location that Cakewalk could use to relocate to. However, there is no additional meaning. Thus, the importation of the MIDI file only allows for a relocation marker.

In the present invention, these meta events are imbued with significantly greater capabilities. These are used to provide navigation, muting, volume, tap-subdivision, etc. capabilities from within the standard MIDI file structure. The programming of this is discussed at '631, 8:66. The extraction process is detailed at '631, 12:17 to 12:32. Types of action classes that can be stored as markers in a Standard MIDI File and extracted at load time can be found at '631, 25:5.

Cakewalk does not anticipate using these markers for this purpose.

D.   Dependent claim 41

Although Cakewalk does support external control and hot keys, it is obvious from the language of the document that each key can only be bound to a single function. Note that on page 214-215 of the Cakewalk owner's manual, any particular key can be assigned to one and only one function. In addition, since there is no provision in Cakewalk for a hotkey map (though provision for hotkeys), there is no way to define a key such that its behavior changes depending on the current location within the sequence or performance.

Note that claim 41 specifically references a plurality of different events to be activated.

Further, it may at first glance appear that the CAL (Cakewalk Application Language) might be able to be used to provide the ability to control multiple events. However, CAL is not designed for realtime control. Note

> CAL is an offline editing and creation tool and cannot execute these real-time commands.
> from http://www.cakewalk.com/Support/FAQ.ASP#23 6-16-2006

E. Dependent claim 42

Although it is possible to change a hot key in Cakewalk by reassigning its key bindings (p214), there is no provision to create a map of these definitions such that the hotkey definition changes during the performance. In fact, there is only one possible key binding per key allowed at a time. Thus, one can not take advantage of a map in order to automate said modification of a hot key. Furthermore, in order to make a change in Cakewalk, one must stop the performance, open the key bindings list, make the change, then save the change, close the page, and then start up the performance again. Thus it is impossible for Cakewalk to perform the claim. Cakewalk does not anticipate these maps, and it can not make a change during a performance.

F. Dependent claim 44

Although it is recognized that Cakewalk can send patch changes to affect an instrument, these patches must be stored directly in the first structure, and these patch changes are then sent directly out of the sequencer without modification.

In the present invention, a patch change within the first data structure can be used as a control value that can then reference the instrument definition file of the second data structure, and the instrument definition file can then be used to send a plurality of different resultant patch

<u>changes</u>. Because the instrument definition file can also draw upon other factors, and because

there can be a large number of possible outcomes, there is no direct one-to-one correlation

between the patch change placed within the first data structure and the resultant set of patch

changes that can be output by the invention. Because these definitions can be changed by

information stored in the instrument map, this can be modified throughout the performance

without modifying the underlying first data structure.

Disclosure of instrument maps can be seen in '631, 18:22 and '631, 13:1.

The loading of instrument definitions is disclosed at '631, 22:25].

G.   Dependent claim 45

Whereas Cakewalk can insert arbitrary labels at any location within the sequence, and

then use these labels for relocation, they are unable to disassociate the underlying measure

information from the sequence. These markers continue to be re-nameable, can exist at any

location including locations not at the top of each measure, and multiple iterations of the label

can exist in the same location. Most importantly, Cakewalk does not use these new values as the

underlying measure representation. By looking at the figure on page 121, one can see that each

marker is still referenced by a location of "measure beat tick". For example "Test Marker #2" is

located at measure 3 beat 2 tick 0. There is no way to modify the underlying measure number

from a sequential list (in other words, there is no such thing as "measure 'Test Marker #2' beat 2

tick 0" for example). Therefore the measure name has not been replaced.

A disclosure about how metric markers are embedded into the standard MIDI file, and

then extracted so that they organically redefine the measure number in the second data structure

can be found at '631, 10:26.

H.   Dependent claim 47

Although Cakewalk can modify data directly within the first data structure, there is no

provision to modify these in real time (during the performance), and there is no way to make

modifications of this information without changing the underlying data. Thus, multiple iterations

of change would continue to corrupt the pristine underlying data. In order to make said change,

the user would have to delete the original material, make a recording run to enter new material

(or input that material manually using a variety of different operations), and then save the new

material. If multiple parts required said modification, then multiple recording runs would have

to be made. This exercise would be impossible during a performance situation.

Because the inertia of an instrument is a dynamic property, the rate of dynamic change in

a particular part can be modified in different ways based on the current inertia value for that

instrument. As the inertia value changes, the actual dynamic values (for example controller 7)

will be modified to represent more subtle or violent variations in the dynamic curve. It would be

impossible for Cakewalk to perform this activity, since in the Cakewalk environment, these

values must be stored explicitly within the instrument track, and then output at playback time

without modification.

In particular, inertia allows a <u>single command or stored value</u> to modify multiple values

within a plurality of instruments. This is also something outside of the scope and unanticipated

by the Cakewalk system.

I.   Dependent claim 50

Whereas Cakewalk can submit external commands, it is unable to provide a dynamic map

that allows for a particular input value to reflect separate external commands based on the

current location within the performance. Once the key has been associated, it is fixed until manually redefined in the key binding editor. In fact, the term "key binding" implies that frequent modification is an anticipated activity, and once set, only rarely would a particular key be re-modified. Re-editing the key bindings is tedious and in fact would be impossible to do during a performance.

In contrast, the present invention allows for an external command map to be created that allows for automated modification of the map depending on the current location within the performance.

The external control map is specified at '631, 18:21.

V. §103 REJECTION BASED ON SU AND HEIDORN

A. Dependent claim 28

For a discussion about how resultant maps can be generated from recording multiple performances, see discussion for dependent claim 28 above.

More specifically towards Heidorn, a resultant map is a composite of multiple maps storing the same parameter, not different or all parameters.

Weighted Average is defined at '631, 19:51.

B. Dependent claims 29,33,34,48,49: A Discussion regarding metric vs rhythmic tap

Before addressing these claims, it is worthwhile to discuss the technical differences between two different approaches to tap tempo.

There are many systems that rely on tap to generate a tempo. However, these systems rely on what is considered "metric tap". Although powerful within themselves, they do not

provide the type of capabilities disclosed in the current application, and in fact do not anticipate these capabilities.

Metric tap determines the tap subdivision based on the current meter. A meter of 4/4 (four quarter notes to the measure) determines that the tap subdivision will be quarter notes, and that there will be four evenly spaced events within the measure. In order to modify the meter, for example if one wanted to tap at eighth note subdivisions, then one would have to redefine the meter as 8/8. This requires modifying the score to reflect this change. One can think of rhythmic tap as the same sort of mental exercise as tapping one's foot along with the music: the tapping is regular and tied to the beat.

No matter what the meter is, there is no way that metric tap can provide multiple tap subdivisions within the same measure. Thus, in metric tap, the performer is forced to provide even subdivisions of the measure, and no changes of subdivision until the meter changes.

Rhythmic tap, on the other hand, determines its tap subdivisions on a rhythmic pattern that can be modified based on a set of data. Although it is possible for rhythmic tap to emulate metric tap (in a 4/4 measure, enter a rhythm of repeating quarter notes), metric tap can not produce the flexibility of rhythmic tap. For example, in a 4/4 measure, metric tap (as discussed above) will force a tap subdivision of four quarter notes. In contrast, rhythmic tap could create a tap pattern subdivision of "quarter-eighth-quarter-eighth-quarter", or "quarter-quarter-half", or "whole", or any number of different patterns or rhythms.

Turning to Heidorn, we discover that the type of tap is clearly defined as metric.

Custom tempos can be ... "tapped" <u>on the beat of the music</u> by the soloist using a keyboard key, footswitch, or some other tapping means." [Heidorn at 9:43].

In fact, Heidorn discusses the exact problem that lead to the development of the present invention's tap patterns (rhythmic tap), but then suggests solutions that indicate that rhythmic tap is not anticipated in the scope of his invention. See, e.g., Heidorn at 9:47, and following. Heidorn proposes a technique of providing interpolation between taps, but there is no mechanism for creating arbitrary subdivisions as recited in, e.g., claim 49.

For a reference to the metric map and the tap subdivision being separate entities, see '631, 18:5 through 18:7. For a discussion about creating different tap subdivisions, see '631, 25:36.

C.   Dependent claim 29

Although Heidorn discusses using accompaniment as play and as tap, there is no mention of being able to move between these particular modes. In fact, this is a feature not present in any sequencer, and one of the major requirements that lead to the current invention. Heidorn specifically declares that the purpose of the tap feature is to record tempo values into a tempo map, and this is done by declaring a section of the music. There is no discussion about how to move between tap and play.

D.   Dependent claim 33

As discussed above, Heidorn ties the tap subdivision to the current meter. There is no provision for making a modification, and no structure within which to store this modification. Thus, Heidorn is unable to generate <u>a map that allows a use to program different tap subdivisions into different portions of the piece</u>.

For a reference about programming for specific tap subdivisions, see '631, 42:19 and

FIG. 38. For more advanced tap subdivision programming by using the advanced editor, see

'631, 43:46 and following, and more specifically, line 54.

E.   Dependent claim 34

Even if Heidorn were able to reprogram an underlying tap subdivision, it would have to

be done by modifying the meter. If the meter is modified, then there is one and only one

possible tap subdivision. There is no provision for overriding the meter, since it is impossible for

the scope of Heidorn to support two different meters in the same measure or location.

The present invention can perform this behavior by defining an external command in the

external command map to be a particular tap subdivision. This has been discussed supra.

F.   Dependent claim 49

As discussed supra, Heidorn relies on the meter to determine where the tap subdivisions

occur. Because of this, there is no way to create a pattern beyond the implied pattern of the

meter and its subdivisions. Heidorn therefore does not anticipate declaring a list of tap patterns,

such that the second data structure can reference these patterns.

Reference to tap patterns can be found at '631, 25:36.

G.   Dependent claim 30

Although Heidorn discloses repeat markers, and also discusses a variety of flavors of

repeat markers, there is no disclosure regarding indefinite repeats. Each of the types of repeats is

based on a particular preordained marker that exists prior to the performance. Since these

markers are placed in the Type 0 MIDI file at a previous time, there is no way to declare a vamp

at any time during the musical performance. In addition, there is no discussion whatsoever about

sending a command multiple times so that the <u>number of times the command is sent determines the number of measures the vamp will enclose</u>.

Heidorn explicitly discusses how these repeats are used. See, e.g., Heidorn at 6:4.

8. Repeat Markers (including D.C. and D.S.)

Markers are typically placed in the sequence at the precise measure, beat and tick that each of these events actually occurs.

There is no provision for placing repeats in the disclosure in any other fashion.

A detailed description of arbitrary vamping can be found at '631, 35:21.

H.   Dependent claim 31

After a close review of Heidorn, no mention can be found of an indefinite loop or vamp. In the last paragraph of column 6 as referenced, the markers defined are <u>repeat</u> (perform a section one more time), <u>multiple endings</u> (the first time, use the first ending material, on the repeat, skip the first ending material and play instead the second ending material, on the third, skip the first and second ending material and play the third ending material, and so on), <u>Da Capo</u> (D.C.: repeat from the beginning to the word *fine*), <u>Dal Segno</u> (note that the inventor misspells this term as D<u>e</u>l Segno: repeat from the sign), and <u>Coda</u> (from *fine*, jump to the coda and play to the end). Each of these explicate a preset and unchangeable number and pattern of repetitions, and therefore there is no indefinite repeat from which to exit.

Although the examiner discusses using a "stop" command as a way to exit a vamp, stopping within a vamp would not be exiting the vamp, since if the play command were sent after stopping, one would still remain in the vamp. Exiting a vamp is a standard technical term in music that is understood by musicians to mean continue playing into the next section.

Applicants are unable to find any reference whatsoever in Heidorn to a "loop until"

function.

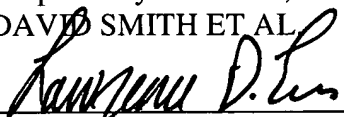A discussion of how exit vamp works can be found at '631, 35:31.

VI.   DOUBLE PATENTING REJECTION

As noted at the beginning of the remarks, a Terminal Disclaimer is being filed

concurrently herewith to obviate the obviousness-type double patenting rejection of claims 52-

64. At least these claims should be in immediate condition for allowance.

In view of the foregoing all of the claims in this case are believed to be in condition for

allowance. Should the Examiner have any questions or determine that any further action is

desirable to place this application in even better condition for issue, the Examiner is encouraged

to telephone applicants' undersigned representative at the number listed below.

PILLSBURY WINTHROP SHAW PITTMAN LLP
1650 Tysons Boulevard
McLean, VA 22102
Tel: 703/770-7900

Date: June 22, 2006

Respectfully submitted,
DAVID SMITH ET AL.

By: _____
Lawrence D. Eisen
Registration No. 41,009

LDE/dkp

Customer No. 00909

400413287v1